Alban Hajdini

CAN-väylään perustuva ohjauspaneelikonseptin suunnittelu

Metropolia Ammattikorkeakoulu

Insinööri (AMK)

Koulutusohjelman nimi

Insinöörityö

8.5.2015

Tekijä(t) Otsikko Sivumäärä Aika	Alban Hajdini CAN-väylään perustuva ohjauspaneelikonseptin suunnittelu 27 sivua + 2 liitettä 8.5.2015
Tutkinto	Insinööri (AMK)
Koulutusohjelma	Auto- ja kuljetustekniikan koulutusohjelma
Suuntautumisvaihtoehto	Tuotetekniikka
Ohjaaja(t)	Lauri Eho

Tämän opinnäytetyön tavoitteena oli suunnitella CAN-väylätekniikkaan perustuva kosketusnäytöllinen ohjauspaneeli. Ohjauspaneelin kokoonpanossa käytetään Arduino-kehitysalustaan perustuvaan mikrokontrollereita sekä 4D Systemsin valmistama kosketusnäyttöä.

Nykyään ajoneuvojen ohjausjärjestelmät ovat älykkäistä yksiköistä ja antureista koostuvia järjestelmiä, joiden tiedonsiirtoliikennöinnissä käytetään CAN-väylätekniikka. Tämän työn yksi keskeisimpiä osa-alueita on CAN-väylätekniikan rakenteeseen ja toimintaan perehtyminen. Työssä tutkittiin CAN-väylän topologia sekä tiedonsiirtorakenteen ominaisuudet ja siihen liittyvät protokollat.

Lisäksi tarkastellaan projektissa käytettyjä Arduino-kehitysalustan perustuvia mikrokontrollereita sekä niiden ohjelmointi C++ ohjelmointikielellä. Työssä kuvataan kuinka mikrokontrollerilla voidaan lukea ajoneuvon tietoja sekä ohjata toimintoja.

Työn alkuperäinen tarkoitus oli, että ohjauspaneelia sovellettaisiin käytettäväksi MoTeC moottorinohjausyksikön kanssa, mutta aiheen vaativuuden sekä kiireisen aikataulun johdosta päädyttiin kehittämään konseptia vain testaustasolle asti, jossa tarkasteltiin ohjelmakoodia sekä väylässä liikkuvaa dataa käyttäen ajoneuvon alkuperäistä moottorinohjausyksikköä.

Avainsanat	CAN-väylä, Arduino, ohjauspaneeli

Author(s) Title Number of Pages Date	Alban Hajdini Designing a of Concept for a CAN-bus-based Control Panel 27 pages + 2 appendices 8 May 2015
Degree	Bachelor of Engineering
Degree Programme	Automotive Engineering
Specialisation option	Automotive Design Engineering
Instructor(s)	Lauri Eho, Lecturer

The aim of this thesis was to design a CAN-bus-technology-based touchscreen control panel. Arduino-based microcontrollers and a 4D System touchscreen were used for the assembly of the control panel.

In today's vehicles, control systems are made up of intelligent units and sensor systems, which use Controller Area Network technology for serial communications. One of the key focusing areas of this thesis was understanding the structure and functioning of the CAN-bus.

This thesis studies the topology of the CAN-bus as well as structural properties of communication and related protocols. In addition, the thesis focuses on the Arduino-platform-based microcontrollers used for this project and their programming with C++ programming language. The thesis examines how vehicle data can be read, and how functions can be controlled with the microcontroller.

The initial aim of the project was to configure the control panel to be used with a MoTeC engine control unit, but due to the complexity of the project and time limitations, a decision was made to work on the concept up to a testing stage. This testing stage consists of developing the program code and reading vehicle data form a vehicle's stock engine control unit.

Keywords	CAN bus, Aurduino, control panel

Sisällys

Lyhenteet

ACK	Acknowledge
CAN	Controller Area Network
CAN-H	CAN-high väyläjohto
CAN-L	CAN-low väyläjohto
CRC	Cyclic Redundancy Check
DLC	Data Length Code
ICSP	In-Circuit Serial Programming
ID	Identifier
ISO	International Standardization Organization
LLC	Logical Link Control
MAC	Medium Access Control
NRZ	Non Return to Zero
Solmu	CAN-väylään liitetty ohjainlaite tai anturi
SAE	Society of Automotive Engineering
SPI	Serial Programming Interface
OSI	Open Systems Interconnect
OBD	On-board Diagnostics

PWM	Pulse-Width Modulation – Pulssinleveysmodulaatio
UART	Universal Asynchronous Receiver Transmitter

1 Johdanto

1.1 Aiheen esittely

Autoissa elektroniikan yleistyminen on ollut erittäin voimakasta viime vuosina. Yhä useampi toiminto tehdään nykyään toimimaan sähköisesti tai mekaanisia toimintoja avustetaan sähköisesti. Myös mukavuusvarusteet ovat lisääntyneet. Nykyään autoihin saa erittäin laajan valikoiman erilaisia mukavuusvarusteita, joiden tarkoitus on helpottaa autoilijan jokapäiväistä elämää. Elektroniikan ansiosta ollaan tultu pitkälle myös auton turvallisuudessa. Nykyään autoissa on ajonvakautus-, lukkiutumattomia jarru- ja monia muita järjestelmiä, jotka ovat tehneet autoilusta yhä turvallisemman. Melkein kaikille näille järjestelmille vaaditaan kuitenkin kytkimiä, jotta niitä voidaan kytkeä päälle ja pois tai säätääkseen niiden toimintaa. Sen seurauksena autojen kojelaudoissa on yhä enemmän kytkimiä erilaisille toiminnoille ja järjestelmille.

Autourheilussa, jossa keskittyminen ajamiseen on tärkeintä, pyritään yksinkertaistamaan kojelaudan layoutia, niin että kuljettajan on helppo käyttää nappeja myös ajon aikana. Tähän tarkoitukseen on olemassa kytkinpaneeliratkaisuja, jossa ajamiseen ja auton toimivuuden kannalta vain oleellisemmat napit sijoitetaan kojelautaan. Kytkinpaneelilla pyritään myös ehkäisemään ylimääräistä johdotusta ja sitä kautta säästämään painoa. Markkinoilla on tarjolla jälkiasenteisia kytkinpaneeleita autourheilun käyttöön. Nämä kytkinpaneelit ovat kuitenkin kalliita ja vaativat suuria muutoksia auton johdinsarjoissa.

Tämän opinnäytetyön idea on suunnitella edullinen kosketusnäytöllinen ohjauspaneeli markkinoilla olevien kytkinpaneeleiden rinnalle. Ohjauspaneelin suunnittelussa otetaan huomioon erityisesti monikäyttöisyys. Tarkoituksena on myös, että ohjauspaneelia on mahdollisuus jatkaa tulevaisuudessa lisäämällä tai muuttamalla sen ominaisuuksia tarpeiden mukaan.

1.2 Työn tavoite

Tämän opinnäytetyön tavoitteena oli suunnitella CAN-väylätekniikkaan perustuvaa kosketusnäytöllinen ohjauspaneeli, jolla ohjataan ajoneuvon erilaisia toimintoja. Opinnäytetyön malliksi otettiin MoTeC-valmisteinen Keypad, joka toimii yhdessä MoTeC moottorinohjauksen kanssa. MoTeC-moottorinohjausyksikkö on autourheiluun tarkoitettu moottorinohjausyksikkö, joka voidaan ohjelmoida omien tarpeiden ja vaatimuksien mukaan. Opinnäytetyön tavoitteena oli rakentaa ohjauspaneeli toimimaan MoTeC moottorinohjauksen kanssa, mutta aiheen vaativuuden sekä kiireisen aikataulun takia päädyttiin kehittämään ohjauspaneelikonseptia vain testaustasolle asti.

Päätavoitteena on selvittää, miten CAN-väylästä pystytään lukemaan tarvittavia tietoja, joita voidaan käyttää hyväkseen ohjauspaneelin komentojen ohjelmoimiseen.

Työn yksi tärkeimmistä osa-alueista on CAN-väylätekniikaan perehtyminen. Nykyään ajoneuvojen ohjausjärjestelmät ovat älykkäistä yksiköistä ja antureista koostuvia järjestelmiä, joiden tiedonsiirtoliikennöinnissä käytetään CAN-väylätekniikka. Koska ohjauspaneelin toiminta perustuu CAN-väylätekniikkaan, on laitteen ohjelmoinnin kannalta tärkeää ymmärtää väylän tiedonsiirtorakennetta ja sen ominaisuuksia.

Laitteen toteutuksessa käytetään hyväkseen opetus- ja harrastelijakäyttöön kehitetty Arduino-kehitysalustaa. Arduino-mikropiirit ovat avoimeen laitteistoon perustuvia mikrokontrollereita ja ohjelmointiympäristö, joiden avulla voi rakentaa ja ohjelmoida ohjainlaitteita eri tarkoituksiin. Työssä keskitytään ohjelmakoodin kirjoittamiseen sekä datan tutkimiseen.

2 CAN-väylä

CAN-väylä (Controller Area Network) on saksalaisen ajoneuvoelektroniikkaan erikoistuneen Robert Bosch GmbH:n vuonna 1983 kehittämä sarjamuotoinen verkkotekniikka ajoneuvoille. CAN toimii kommunikointiväylänä ajoneuvossa erilaisten ohjausjärjestelmien välillä. Alun perin väylä suunniteltiin käytettäväksi ainoastaan ajoneuvoväyläksi, mutta CAN-väylän vikasiedettävyyden ja joustavuuden ansioista ne ovat käytössä erittäin laajasti kaikenlaisissa automaatiojärjestelmissä. [1, s. 14–15.]

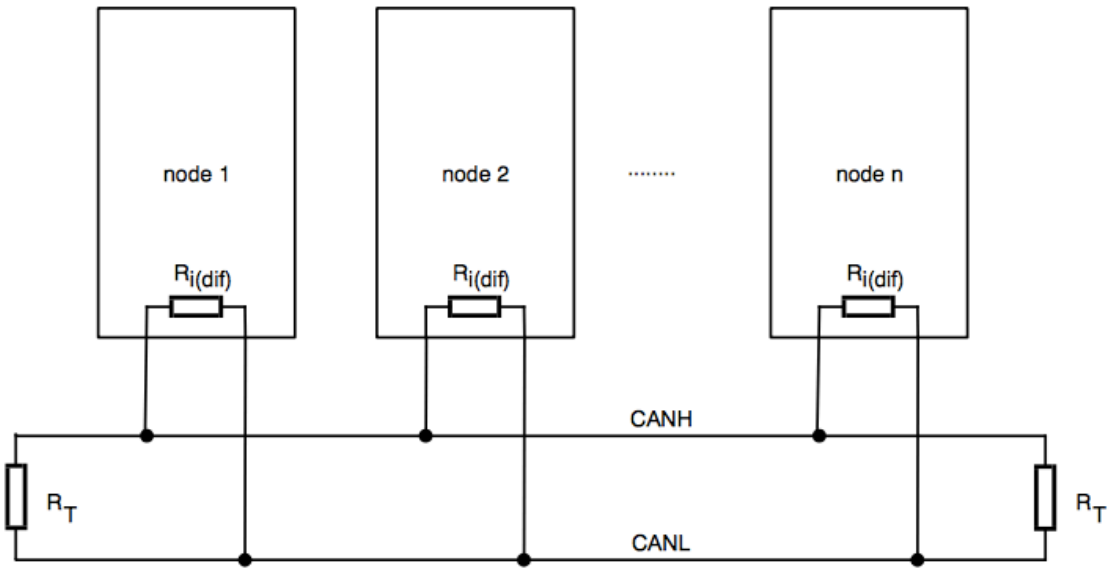
CAN-väylä on rakenteeltaan melko yksinkertainen. Varsinainen väylä koostuu kierretystä parikaapelista, joka kulkee jokaisen ns. solmun kautta ja se päätetään päätevastuksilla. Päätevastukset ovat tyypillisesti arvoltaan 120 Ω suuruisia, ja niiden tarkoitus on vaimentaa mahdollisia sähkömagneettisia heijastuksia. [2, s. 5–6.]

Väylän tiedonsiirtonopeus määräytyy käytettävän kaapelin pituuden mukaan. Väylän tiedonsiirtonopeus rajoittuu 10 kbit/s ja 1 Mbti/s:n välille. Käytettäessä tiedonsiirtonopeutta 1 Mbti/s, saa väylän pituus olla maksimissaan 40 m. Hitaammalla tiedonsiirtonopeudella 10 kbit/s voi väylän pituutta kasvattaa jopa 6 000 metriin saakka. Väylän enimmäispituutta ei voida kasvattaa, ilman että tiedonsiirtonopeutta alennetaan, sillä rajana tulee vastaan sähkömagneettisen aallon kulkunopeus. Jotta solmut ehtisivät ottaa näytteen yksittäisestä bitistä reaaliajassa, on tärkeää, että siirtotien viive ei ole liian suuri. [2, s. 4–5.]

Can-väylä perustuu ns. multi-master-periaatteeseen (kuva 1), jossa väylä kulkee jokaisen solmun kautta ja jossa kaikilla solmuilla on yhtä suuri oikeus lähettää viestinsä väylälle. Tämä tarkoittaa, että jokainen solmu voi oma-aloitteisesti lähettää sanoman väylälle yleisesti vastaanotettavaksi. Yhdessä sanomassa voi lähettää korkeintaan 8 tavua eli 64 bittiä tietoa. Sanoma sisältää tunnistekentän (Identifier [ID]), joka määräytyy sanoman sisällön mukaan ja vain sanoman sisältämän tiedon tarvitsevat solmut ottavat sen vastaan. [2, s. 5–6.]

CAN-solmu koostuu kolmesta yksiköstä: keskusyksikkö, CAN-ohjain ja lähetin-vastaanotinyksikkö. Keskusyksikkö käsittelee vastaanotetut viestit ja päättää mitä viestejä se haluaa lähettää väylälle. CAN-ohjain on yleensä integroitu keskusyksikköön, ja sen tehtävä on purkaa viestit keskusyksikölle luettavaksi tai jos viestejä lähetetään, niin

se muuttaa ne sarjamuotoon. Lähetin-vastaanotinyksikkö vastaa nimensä mukaan viestien lähettämisestä ja vastaanottamisesta. [1, s. 29.]



Kuva 1. Periaatekuva lineaarisesta CAN-väylästä [10].

2.1 ISO 11898 -standardi

CAN-väylä on standardisoitu ISO:n OSI-mallin mukaan. Verkkoprotokollat määritellään yleensä kerroksissa, jotka käsittävät väylän ominaisuuksia ja tehtäviä. OSI-malli on jaettu 7:ään eri kerrokseen, joista CAN-spesifikaatio täyttää näistä kaksi: fyysisen- ja siirtoyhteyskerroksen. [1, s. 26–27.]

- Fyysinen kerros käsittää mm. jännitetasot, signaloinnit ja tuetut siirtonopeudet.
- Siirtoyhteyskerros käsittää mm. tietokehyksen, virheiden tunnistaminen ja niihin reagoimisen sekä viestien prioriteetin määrittely. [4, s. 7.]

Alkuperäinen CAN-väylä standardi ISO 11898 määritteli vain osittain vaatimuksia fyysiselle ja siirtoyhteyskerrokselle. Nykyään ISO 11898 -standardi on jaettu moneen osaan, joista ISO 11898-1 ja ISO 11898-2 määrittelevät high speed CAN-väylän ja ISO 11898-3 määrittele low speed- tai fault tolerant- CAN-väylää. [3, s. 2.]

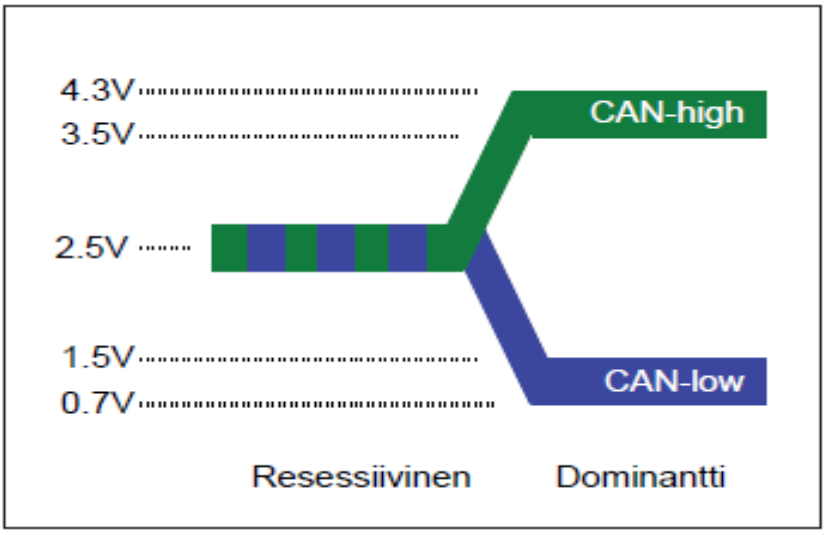
CAN-väylälle on määritelty myös sovellusaluekohtaisia standardeja, jotta valmistajien väliset laitteet olisivat yhteensopivia. SAE:n kaksi tärkeintä henkilöautoihin sovellettavaan CAN-väylästandardia ovat SAE J2284 ja SAE J2411. SAE J2284 -standardi määrittää high speed -CAN-väylää, joka on tarkoitettu korkeille tiedonsiirtonopeuksille henkilöautoissa ja perustuu ISO 11898-2 standardiin. SAE J2411 –standardi määrittää single wire -CAN-väylää eli henkilöautojen yksijodinväylän. [2, s. 4.]

2.2 Fyysinen kerros

Fyysinen kerros määrittele tavan, jolla signaalia lähetetään, eli sen rooli on varmistaa bittien fyysisen siirron solmujen välillä. Määritelmän mukaan fyysinen kerros pitää sisällään seuraavat osa-alueet: fyysinen signaalinkäsittely, ajoituksen synkronointi sekä fyysisen siirtovälineen (väylä, solmut ja väyläliittimet) ominaisuudet. [1, s. 27–29.]

2.2.1 High speed CAN

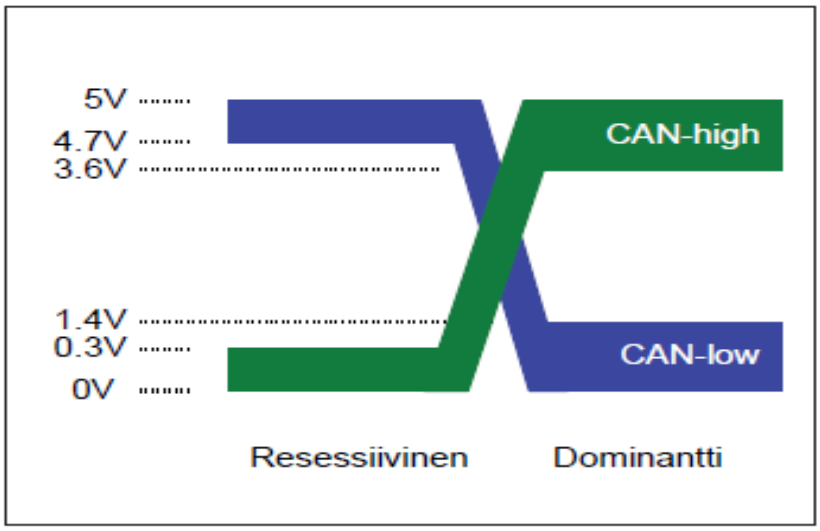
CAN-väylässä parikaapelijohtimet on nimetty CAN-High ja CAN-Low. Väylän toiminta perustuu näiden kahden johtimen jännite-eroon. Väylässä vallitseva jännite-ero määrää väylän ns. loogisen tilan. Loogisia tiloja on kaksi: dominantti, joka vastaa loogista tilaa 0, ja resessiivinen, joka vastaa loogista tilaa 1. Kun CAN-High-johtimessa jännite on 3,5 V ja CAN-Low-johtimessa jännite on 1,5 V, silloin puhutaan dominanttitilasta ja jännite-ero johtimien välillä on silloin 2 V. Resessiivisessä tilassa molempien johtimien jännite on 2,5 V, jolloin jännite-ero silloin on 0 V. [4, s. 8.]



Kuva 2. ISO 11898-2:n mukaan määritelty jännitetasot [4, s. 8].

2.2.2 Fault-tolerant CAN

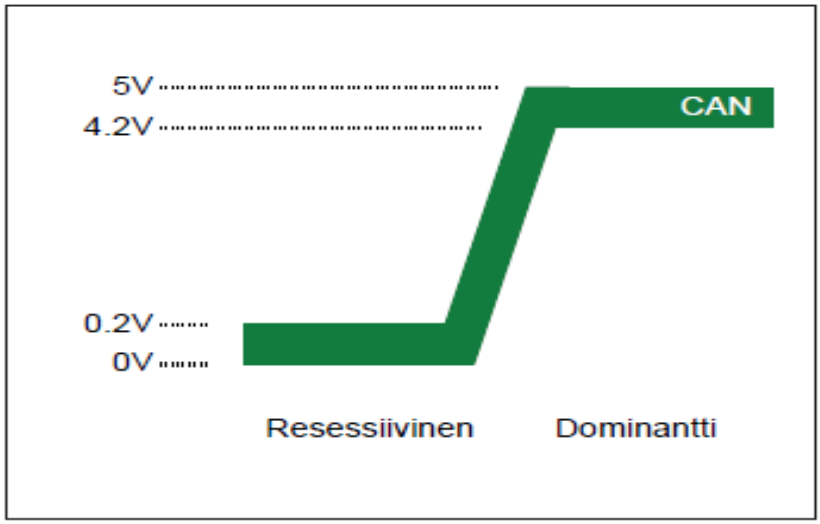
Fault-tolerant CAN, joka tunnetaan myös nimellä low Speed CAN, on matalille siirtonopeuksille tarkoitettu väylä. Väylän toimintaperiaate ei eroaa paljon High speed CAN:sta, mutta sen ominaisuuksiin kuuluu, että se voi toimia hetkellisesti pelkästään CAN-High'n tai CAN-Low'n avulla. Tämä on saavutettu kasvattamalla väyläsignaalien resessiivisten ja dominanttien tasojen välistä minimijännite-eroa. Suuremmista jännite-eroista johtuen väylän siirtonopeutta on jouduttu rajoittamaan nopeuteen 125 kbit/s. [4, s. 8.] Kuva 3 havainnollistaa jännite-erot.



Kuva 3. ISO 11898-3:n mukaan määritelty jännitetasot [4, s. 8].

2.2.3 Single wire CAN

Single wire eli yksijohdinväylä käyttää ainoastaan yhtä johdinta, CAN-High'ta, jonka kautta tiedonsiirto tapahtuu. Toisin kuin muissa väyläkonfiguraatioissa, single wire-väylän toiminta perustuu jännitetasojen mittaukseen, eikä jännite-ero mittaukseen. [4, s. 8.]



Kuva 4. SAE J2411:n mukaan määritelty jännitetaso [4, s. 8].

2.3 Siirtoyhteyskerros

Siirtoyhteyskerroksessa määritellään viestitettävän tiedon luontia ja käsittelyä. Siirtokehyskerros luo rakennekehykset sekä priorisointi määritelmän viesteille, joiden mukaan tiedonsiirto tapahtuu väylässä. OSI-mallin mukaan siirtoyhteyskerros jaetaan kahteen alikerrokseen, jotka määrittelevät tiedon rakennetta ja käsittelyä; MAC (Medium Access Control) ja LLC (Logical Link Control). [1, s. 28.]

MAC-alikerroksen määrittelemät toiminnot ovat seuraavat:

- tiedon pakkaus viestikehyksiin/-kehyksistä
- viestikehysten koodaus/purku (bit-stuffing)
- virheiden tunnistaminen
- virhetiedon välitys

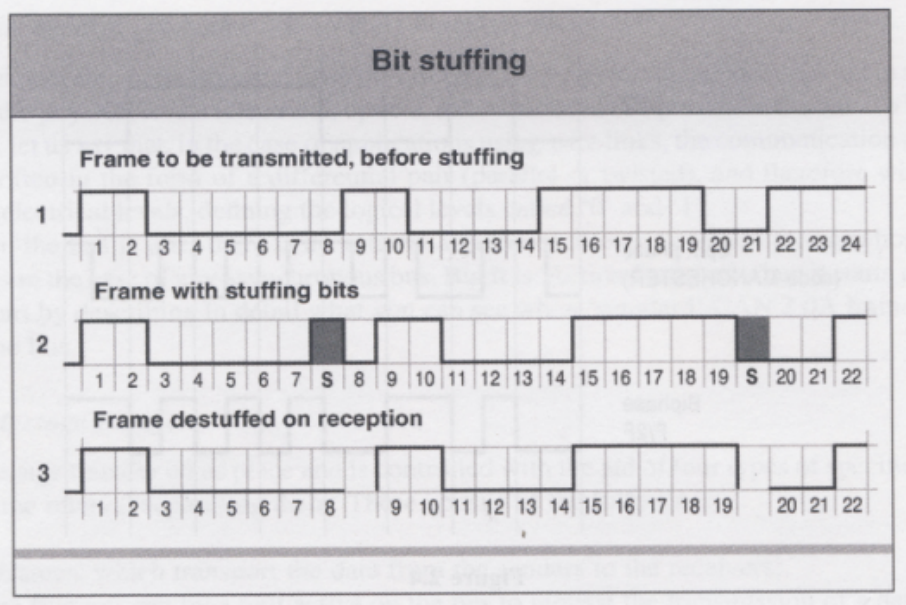
- viestikehysten kuittaus.

LLC-alikerroksen määrittelemät toiminnot ovat

- viestikehysten suodatus
- ylikuormituksen ilmaisu
- virheistä palautuminen. [4, s. 8.]

2.4 NRZ-koodaus ja Bit Stuffing-menetelmä

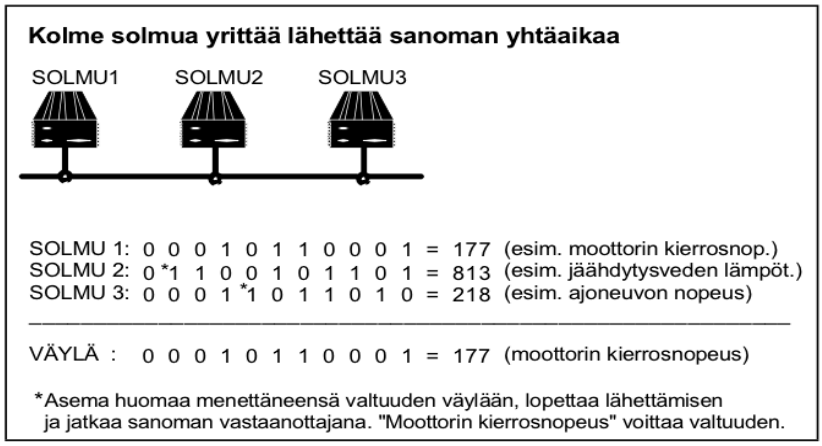
Väylässä liikkuva signaali koodataan NRZ (Non Retrurn to Zero) -koodausmenetelmällä binaarisiin jännitetasoihin. (Jännitetasoilla määritettiin väylälle ns. loogisia tiloja, resessiivinen tila (1) ja dominantti tila (0)). Solmujen lähettämät sanomat sisältävät monta samanarvoista bittiä peräkkäin ja koska bittien erottelu perustuu kestoaikaan, voivat sanoman vastaanottavat solmut tulkita tilanteen virheelliseksi ja kommunikaatio pysähtyy. Bit Stuffing -menetelmällä varmistetaan solmujen välistä synkronointia sijoittamalla viiden peräkkäisen bitin jälkeen yksi vastakkainen ”stuff-bitti”. Sanoman vastaanottanut solmu sitten kääntää tämän ”stuff-bitin” takaisin alkuperäiseen muotoon ja varmistaa näin viestin eheyden. (kuva 5) [1, s. 37–38.]



Kuva 5. Bit Stuffing-menetelmä [1, s. 38].

2.5 Kilpavaraus

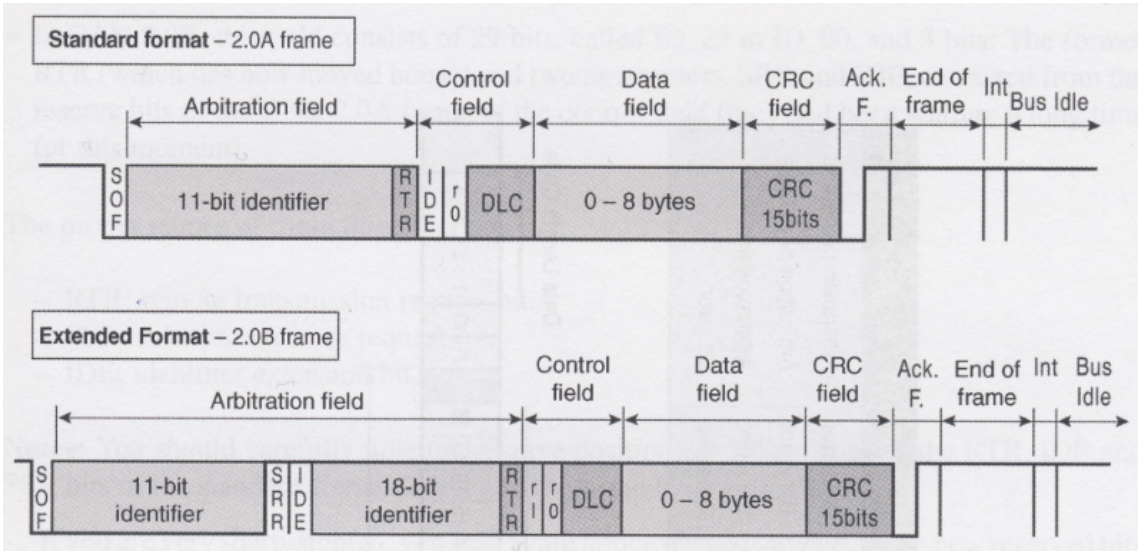
Väylälle lähetetyt sanomat priorisoidaan käyttämällä kilpavarausmenetelmää. Siis kun useampi solmu yrittää saman aikaisesti lähettää sanoman väylälle, ratkaistaan lähetysvuoro tunnistekentän avulla. Tunnistekentältään pienin arvo saa väylässä suurimman prioriteetin ja päinvastoin. [2, s. 7.] Kuvassa 6 on esitetty esimerkkitilanne kilpavarauksesta.



Kuva 6. Kilpavarauksen periaate [2, s. 7].

2.6 Standard CAN ja Extended CAN

Standard CAN ja Extended CAN (kuva 7) on määrittely Boschin CAN -spesifikaatiossa CAN versioiden CAN 2.0A ja CAN 2.0B alla. Eroavaisuus näiden kahden välillä näkyy niiden tunnistekentissä. Standard CAN-formaatissa tunnistekenttä on 11 bittiä, ja Extended CAN'ssa tunnistekenttä on 29 bittiä pitkä. Extended CAN -formaatissa RTR-bitti on siirtynyt tunnistekentän perälle ja tilalle on tullut SRR-bitti (Substitute Remote Request). Väylä tukee molempia formaatteja, mutta resessiivisen SRR-bitin ansiosta Standard CAN-formaatilla on suurempi prioriteetti. [1, s. 76–79.]



Kuva 7. Kuva havainnollistaa Standard CAN -formaatin ja Extended CAN -formaatin erot [1, s. 77].

2.7 Viestin kehysrakenne

CAN-protokolla määrittelee neljä erilaista viestikehystä:

- sanomakehys – datakehys
- kyselykehys – RTR-kehys
- virhekehys
- viivekehys [4, s. 9].

2.7.1 Sanomakehys – Datakehys

Sanomakehys (kuva 8) esittää tiedonsiirtorakennetta, jolla solmut kommunikoivat keskenään. Sanomakehys voidaan jakaa kahdeksaan eri pääosaan, joita kutsutaan ”kentiksi”.

- SOF (Start of Frame): Koostuu yhdestä dominantti bitistä ja signaloi viestikehyksen aloituksen. Ennen kuin tiedon vaihto voi alkaa, täytyy väylän olla vapaana. Aloitusbitti käytetään myös apuna solmujen synkronointiin.
- Sovittelukenttä (Arbitration field): Koostuu kahdesta osasta; tunnistekentästä (ID (Identifier)) ja RTR (Remote Transmission Request)-bitistä. Standard Can-muodossa tunnistekenttä on 11 bittiä ja Extended Can -

muodssa 29 bittiä (kuva 7). Tunnistekentässä jokaista viestikehystä yksilöidään, ja eniten merkitsevä bitti lähetetään aina ensin. Sanomakehyksessä RTR-bitin on aina oltava dominantti.

- Ohjauskenttä (Control Field): Kuusi bittiä pitkä. Sisältää IDE (Identifier Extension)-bitin, joka määrittää tunnistekentän pituuden. IDE-bitin olleessa dominantti, tunnistekenttä on 11 bittiä ja IDE-bitin olleessa resessiivinen on tunnistekenttä 29 bittiä. IDE-bitin jälkeen tulee yksi varalla oleva bitti, jonka tulee olla dominantti. Loput neljä bittiä ovat DLC (Data Length Code)-kenttä ja sillä määritettään tietokentän tavujen määrää.

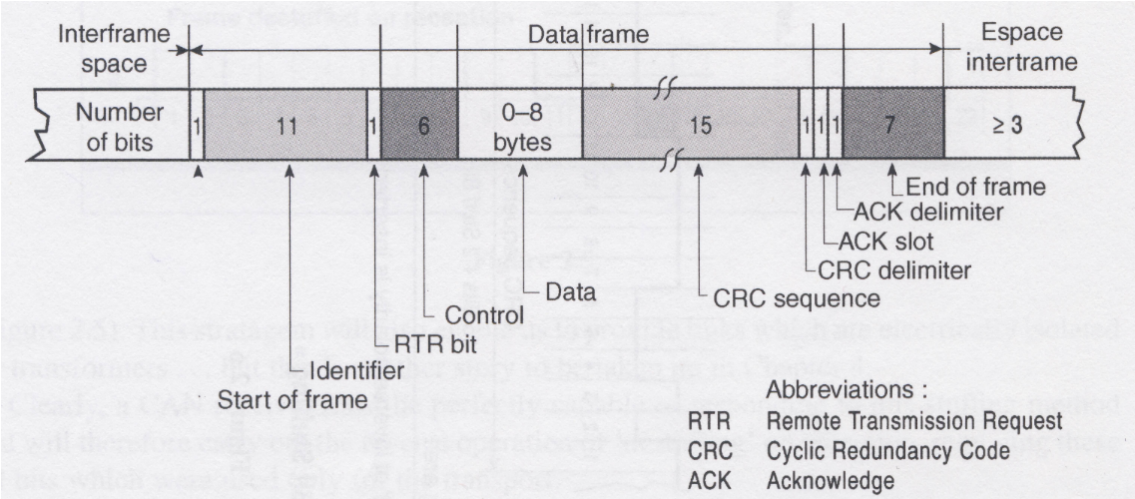
- Tietokenttä (Data Frame): Kenttä jossa lähetettävä tieto sijaitsee. On enintään 8-tavun mittainen ja maksimissaan 64 bittiä. Eniten merkitsevä bitti lähetetään ensin.

- CRC (Cyclic Redundancy Code): Koostuu 15 bitin tarkistus-sekvensisstä ja yhdestä lopussa olevasta CRC-erotinbitistä.

- Kuittauskenttä (Acknowledgement field): Kahdesta bitistä muodostuva kuittauskenttä (ACK slot ja ACK delimiter). ACK slot:n avulla solmut kuittaavat ottaneensa sanoman vastaan ja kirjoittavat tähän dominantin bitin. Erotinbitti (ACK delimiter) tulee aina olla resessiivinen, jolla varmistetaan lähetyksen onnistuneen ilman virheitä.

- Lopetuskenttä (End Of Frame): Lopetuskenttä koostuu seitsemästä resessiivisestä bitistä.

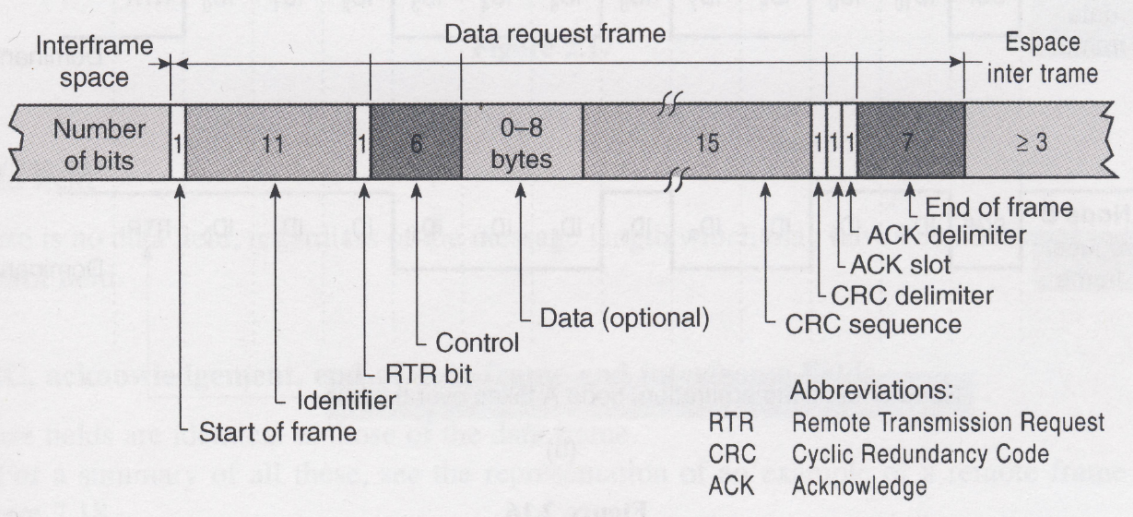
- IFS (Interframe Space): Koostuu kolmesta resessiivisestä bitistä, jotka erottavat viestikehykset toisistaan. [1, s. 40–54.]



Kuva 8. Sanomakehyksen rakenne (Standard CAN-formaatissa) [1, s. 40].

2.7.2 Kyselykehys – RTR-kehys

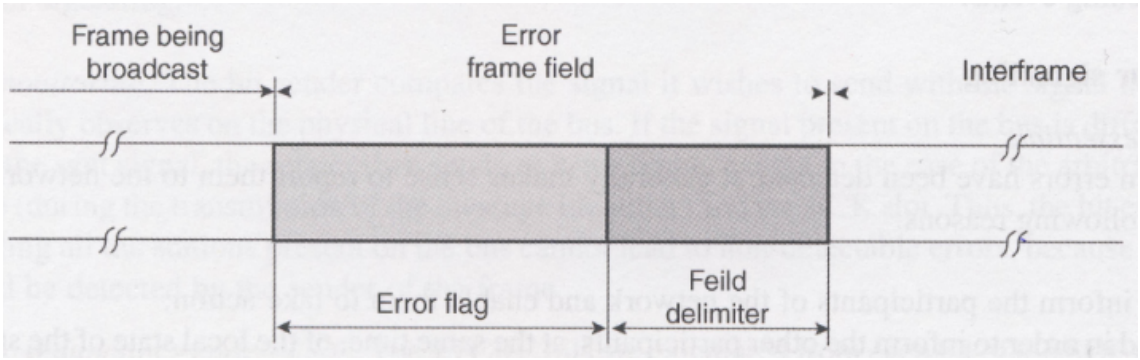
Kun jokin solmu tarvitse jotain tiettyä tietoa, se voi tehdä kyselyaloitteen lähettämällä kyselykehyksen. Kyselykehyksen tietokentällä ei ole datatavuja, vaan sen ohjauskenttä kuvaa vastaavan sanomakehyksen sisältämien datatavujen määrän. Kyselykehyksen pääeroavaisuus sanomakehyksestä määräytyy RTR-bitin mukaan. Kyselykehyksessä RTR-bitti on resessiivinen. [1, s. 51.]



Kuva 9. Kyselykehyksen rakenne. [1, s.51]

2.7.3 Virhekehys

Jos väylällä esiintyy virheellinen viesti, virheen havainnut solmu lähettää virhekehyksen väylälle. Kun muut solmut ovat myös havainneet virhekehyksen väylällä, ne keskeyttävät meneillään olevien sanomien lähettämisen ja alkavat sen sijaan itsekin lähettämään virhekehyksen. Virhekehyksiä on kahdentyyppisiä; aktiivinen ja passiivinen virhekehys. Aktiivinen virhekehys koostuu kahdesta osasta, virhelipusta ja erotinkentästä. Virhelippu koostuu kuudesta peräkkäisestä dominantista bitistä ja erotuskenttä koostuu kahdeksasta resessiivisestä bitistä. Passiivinen virhekehys on rakenteeltaan samanlainen kuin aktiivinen; erona on virelippu joka koostuu kuudesta peräkkäisestä dominantista bitistä. [1, s. 65–68.]



Kuva 10. Virhekehyksen rakenne. [1, s. 66].

2.7.4 Viivekehys

Viivekehys on rakenteeltaan samanlainen kuin virhekehys. Viivekehys koostuu kahdesta osasta, viivelipusta ja erotinkentästä. Viivelipussa on kuusi peräkkäistä dominanttibittiä ja erottimessa on kahdeksan resessiivibittiä. Viivekehyksellä solmu varaa itselleen väylän edellisen vastaanotetun viestin käsittelyn ajan. Jotta väylää ei estetä kokonaan, solmut voivat pitää väylän varattuna ainoastaan kahden peräkkäisen viivekehyksen verran. [1, s. 71–74.]

2.8 Vianhallinta

CAN-väylässä vikoja pyritään ennaltaehkäisemään erilaisilla sisäänrakennetuilla parametreilla kuten esimerkiksi virheentarkastus, Bit Stuffing tai pelkästään tarkastamalla yksittäisiä bittejä. Ennaltaehkäisyn lisäksi vikojen esiintyessä systeemiin on rakennettu erilaisia vianhallintaprosesseja, joilla varmistetaan väylän toimivuutta. [1, s. 54–56.]

Virheentarkastuksella varmistetaan, että väylän kautta lähetetyt viestit saapuvat vastaanottajalle muuttumattomana. Virheidentarkistus tapahtuu mm. 15-bittisen CRC-sekvenssin avulla. CRC-sekvenssissä datasta muodostetaan tiivistealgoritmilla vakiokokoinen luku. Viestin vastaanottanut solmu tarkistaa datan oikeellisuutta laskemalla tämän tarkistussumman uudelleen. Jos summa ei täsmää alkuperäisen kanssa, luokitellaan viesti virheelliseksi. [1, s. 44–45.]

Kun väylässä kuitenkin esiintyy virheitä, on tärkeää tietää, millaisesta virheestä on kyse: onko se harvoin esiintyvä virhe ja näin ollen ”ei niin vakava” vai onko kyseessä jat-

kuvasti esiintyvästä virheestä, jolloin se luokitellaan ”vakavaksi”. Jotta väylässä esiintyvien virheiden vakavuus voidaan määrittää, on solmuihin sisäänrakennettu virhelaskuri sekä lähetys- että vastaanottovirheille. Virhelaskuri toimi, niin että virheellisesti lähetetyt ja vastaanotetut viestit kasvattavat laskurin arvoa, ja päinvastoin virheettömät viestit laskevat virhelaskurin arvoa. (Huom. virhelaskurin laskutapa ei ole suoran verrannollinen onnistuneiden ja virheellisten viestien välillä.). [1, s. 56–57.]

Jos solmun virhelaskurin arvo on välillä 0 ja 127, se toimii ns. aktiivisessa virhetilassa. Tässä tilassa solmu lähettää aktiivisen virhekehyksen havaittuaan virheen väylässä. Aktiivinen virhekehys signaloi muille solmuille virheestä väylällä, jolloin ne keskeyttävät meneillään olevien sanomien lähettämisen ja aloittavat sen sijaan itsekin lähettämään virhekehyksen. [1, s. 57.]

Kun arvo on välillä 128 ja 255, solmu on silloin passiivisessa virhetilassa. Tässä tilassa solmu lähettää passiivisen virhekehyksen havaittuaan virheen väylässä. Passiivinen virhekehys koostuu vain resessiivisistä biteistä, jolloin väylän toiminta ei häiriytyy enää kyseisestä solmusta. [1, s. 57–59.]

Jos virhelaskuri ylittää arvoa 255, se ei enää voi lähettää eikä vastaanottaa viestejä ja siirtyy ns. bus off-tilaan, eli solmu poistuu automaattisesti väylältä. Bus off –tilassa solmu resetoituu ja voi palata takaisin väylälle nollatun virhelaskurin kanssa. [1, s. 59.]

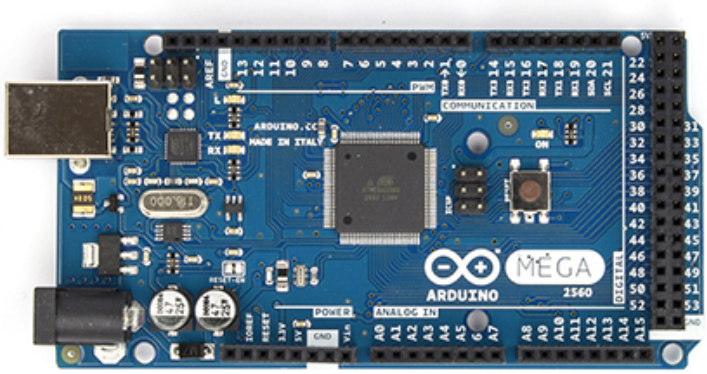
3 Ohjauspaneeli

Ohjauspaneeli koostuu kolmesta pääkomponentista: mikrokontrolleri, CAN-lisälaite ja kosketusnäyttö. Mikrokontrolleri toimii laitteen keskusyksikkönä. CAN-lisälaite, joka pitää sisällään CAN-ohjainpiirin ja CAN-vastaanotin-lähetinpiirin, toimii tiedonsiirtosiltana keskusyksikön ja CAN-väylän välillä, ja kosketusnäytöllä ohjataan toivottuja toimintoja.

Projektissa käytetään Arduinon avoimeen laitteistoon perustuvaa mikrokontrollerialustaa sekä 4D Systemsin valmistamaa kosketusnäyttöä. Arduino perustuu 8-bittiseen Atmel AVR -mikrokontrolleriin. Piirilevyissä on sarja digitaalisia ja analogisia I/O-liitäntäpinnejä, joihin voidaan kytkeä erilaisia laajennuskortteja sekä muita piirejä. 4D Systemsin valmistama, DIABLO16-prosessorilla varustettu 7,0”:n kosketusnäyttö toimii laitteen ohjaustyökaluna, jolla suoritetaan halutut komennot. Näyttö on yhteen sopiva Arduino-alustojen kanssa. [5 ; 9]

3.1 Arduino Mega 2560

Arduino Mega 2560 (kuva 11) on Atmel ATmega2560 -mikroprosessoriin perustuva mikrokontrolleri, joka toimii 16 MHz:n kellotaajuudella. Mega 2560:ssa on 54 digitaalista I/O-pinniä, 16 analogista sisääntuloa ja 5 V:n ja 3,5 V:n pinnit. Digitaalisista I/O-pinneistä 15 toimii myös 8-bittisinä PWM-lähtöinä (Pulse-Width Modulation). Ohjainpiirissä on myös neljä UART-sarjaporttia, ICSP-portti ja reset-painike. Mega 2560:ssa on sekä USB-liitin että DC-virtaliitin, jotka molemmat voidaan käyttää virtalähteinä. USB-liitännällä Mega 2560 toimii 5 V:n käyttöjännitteellä. Käytettäessä erillistä virtalähdettä, suositusrajat ovat 7–12 V. Alustassa on 257 kB Flash-muistia, 8 kB SRAM-muistia ja 4 kB EEPROM-muistia. [5]



Kuva 11. Arduino Mega 2560 -mikrokontrolleri [5].

3.2 Seeed Studio CAN Bus Shield

Seeed Studion CAN Bus Shield on lisälaite, joka perustuu myös Arduino-alustalle. CAN Bus Shield toimii yhdistettynä Arduino Mega 2560 -mikrokontrollerin kanssa CAN-solmuna. Ohjain käyttää MCP2515 CAN Bus -piiriä SPI-liitännällä ja MCP2551 CAN -lähetin-vastaanotinpiiriä, joilla se antaa Arduino Megalle CAN-väylä valmiudet. Ohjain tukee sekä Standard Frame- että Extended Frame -formaatteja. Se on myös varustettu OBD-II adapterilla, joka voidaan kytkeä auton diagnostiikkapistokkeeseen. [8]

Kuva 12. Seed Studion CAN Bus Shield [8].

3.3 7.0”:n Intelligent Display Module – Touchscreen

uLCD-70DT on 4D Systemsin kehittämä kosketusnäyttö (kuva 13), joka käyttää DIAB-LO16-prosessoria. Näytössä on 16 I/O-pinniä, joista 4 voidaan käyttää analogiseen sisääntuloon. Näytössä on myös micro-SD-lukulaite, jonka avulla näyttöön voidaan tuoda graafisia suunnitelmia. Arduino Display Shieldi -adapterin avulla näyttö voidaan kytkeä helposti muihin Arduino-alustoihin. [9]

Kuva 13. 4D Systemsin uLCD-70DT-kosketusnäyttö sekä Arduino-adapteri [9].

3.4 Ohjelmointiympäristöt

3.4.1 Arduino IDE

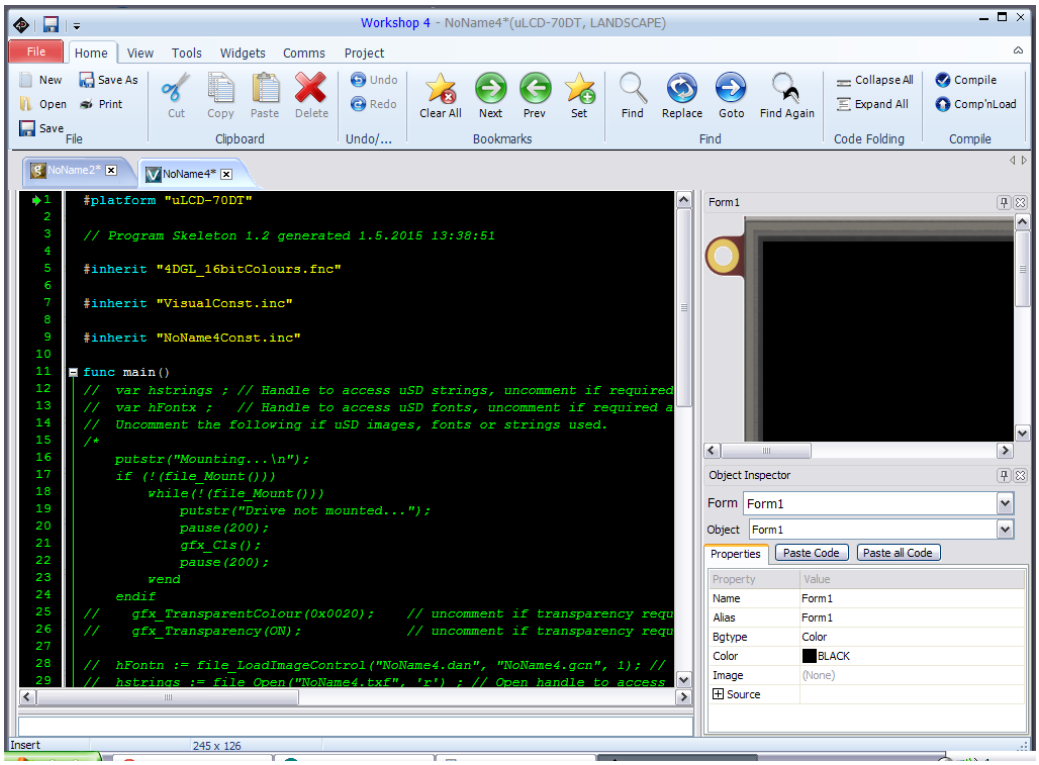
Arduinolla on oma ohjelmointiympäristönsä, jota kutsutaan Arduino IDE:ksi (kuva 14) Arduino-laitteistoa ohjelmoidaan C++:aan perustuvalla Arduino-ohjelmointikielellä. Koska Arduino-laitteet perustuvat Atmel ATmega mikrokontrollereihin, sallii Arduino myös AVR C-ohjelmointikielen käyttämisen. IDE-ohjelmointiympäristöllä koodi kirjoitetaan suoraan ohjelmointi-ikkunaan, josta se ladataan Arduino-laitteelle USB-väylän kautta. Ohjelmointiympäristöön on valmiiksi tallennettu koodikirjasto, josta löytyy ohjelmakoodi erilaisille laitteille ja erillisiin tarkoituksiin. Koodikirjastoon voi myös tallentaa omat koodiluonnokset. [6]

Kuva 14. Kuvakaappaus Arduino IDE:stä.

3.4.2 4D Systems Workshop IDE

4D Systemsin Workshop4 on näyttömoduulien ohjelmointiin tarkoitettu IDE-ohjemistoympäristö. Workshop4 sisältää neljä kehitysympäristöä: Designer, ViSI, ViSi – Genie ja Serial, joiden avulla voi luoda graafisia luonnoksia, kirjoittaa koodia ja hallita laitteistoa. Designer-kehitysympäristöllä ohjelmoidaan näyttömoduuleja käyttäen 4DGL-koodia. 4DGL on grafiikkaan suuntautunut ohjelmointikieli, joka mahdollistaa nopeat ja monimuotoiset sovelluskehitykset. ViSi-kehitysympäristöllä luodaan graafisia luonnoksia. Sen drag-and-drop –toiminnolla näytölle voidaan siirtää kohteita, joka mahdollistavaa helpon suunnittelun. ViSi-Genie on uusin ja kehittynein 4D Systemsin kehitysympäristö. ViSi-Genie:llä voi tehdä samat toiminnot kuin ViSi:llä; erona on, että se muuttaa graafiset luonnokset automaattisesti lähdekoodiksi. Tämä toiminto helpottaa huomattavasti näyttöjen ohjelmoimisen, eikä koodia tarvitse itse kirjoittaa. Serial-

kehitysympäristöllä näytöstä voi tehdä orjamoduulin, jonka voi sitten ohjata maser-mikrokontrollerilla sarjaportin kautta. [9]



Kuva 15. Kuvakaappaus Workshop4 -ohjelmointiympäristöstä.

4 Ohjelmointi

4.1 CAN-solmun ohjelmointi

Arduino Mega 2560 ja CAN Bus Shield muodostavat yhdessä CAN-solmun. Jotta solmulla voidaan lukea CAN-väylässä kulkevia sanomia sekä lähettää komentoja, tarvitaan ohjelmakoodi, jolla määritetään CAN-protokollan mukaiset parametrit. Seeed Studion CAN-Bus Shieldille on olemassa looveen kirjoittamia ohjelmakoodiesimerkkikirjastoja, jotka käytetään hyväksi CAN-solmun ohjelmoinnissa. Koodikirjaston lisäksi sivustolla on myös ohjeet erilaisille komennoille. Koodikirjastot löytyvät osoitteesta https://github.com/Seeed-Studio/CAN_BUS_Shield. [8]

4.1.1 Tiedonsiirtonopeus

Ajoneuvon väylän käyttämä tiedonsiirtonopeus (Baud Rate) on ensin selvitettävä, jotta CAN-solmu voidaan ohjelmoida käyttämään samaa siirtonopeutta. Siirtonopeudet voivat olla väliltä 5 kbit/s ja 1Mbit/s. [9] Kuvassa 11 on esitetty komennot eri siirtonopeuksille.

```
#define CAN_5KBPS    1
#define CAN_10KBPS   2
#define CAN_20KBPS   3
#define CAN_31K25BPS 4
#define CAN_33KBPS   5
#define CAN_40KBPS   6
#define CAN_50KBPS   7
#define CAN_80KBPS   8
#define CAN_95KBPS   9
#define CAN_100KBPS  10
#define CAN_125KBPS  11
#define CAN_200KBPS  12
#define CAN_250KBPS  13
#define CAN_500KBPS  14
#define CAN_1000KBPS 15
```

Kuva 16. Komennot siirtonopeuden asettamiselle [9]

4.1.2 Maski- ja suodatinrekisterit

Solmut lähettävät sanomiaan väylälle yleisesti vastaanotettavaksi. Sanomat sisältävät tunnistekentän, ja vain sanoman sisältämän tiedon tarvitsevat solmut ottavat sen vastaan. [2, s. 5.]

Maskit ja suodattimet -komennoilla rajataan sanomia tunnistekentän perusteella. MCP2515-piiri sisältää kaksi maski- ja kuusi suodatinrekisteriä, jotka takaavat datan saannin kohdelaitteelta:

```
init_Mask(unsigned char num, unsigned char ext, unsigned
char ulData);
```

, ja

```
init Filt(unsigned char num, unsigned char ext, unsigned
char ulData);
```

num kuvaa, mitä rekisteriä käytetään. Maskeille 0 tai 1, ja suodattimille 0–5.

ext kuvaa sanomakehystyyppiä. Standard Frame -formaatissa käytetään 0:aa ja Extended Frame -formaatissa käytetään 1.

ulData kuvaa maski- tai suodatinrekisterin sisällön eli sanoman sisältämiä datatavuja. [8]

4.1.3 Kuittauskomento

Kuittauskomennolla MCP2515-mikrokontrolleri voi toimia joko kiertokyselytilassa, jossa ohjelmisto tarkistaa onko se vastaanottanut sanoman, tai käyttämällä muita pinnejä viestittämään onko sanoma vastaanotettu tai siirto suoritettu. Seuraavalla komennolla kuitataan vastaanotettuja sanomia:

```
INT8U MCP_CAN::checkReceive(void);
```

Komento palautuu 1:ksi, jos sanoma on vastaanotettu tai siirto on suoritettu. Komento palautuu 0:ksi, jos mitään ei ole vastaanotettu. [8]

4.1.4 Tunnistekentän selvittäminen

Kun vastaanotetaan dataa, on tärkeää selvittää, mikä solmu sen on lähettänyt. Seuraavalla komennolla selvitetään ”lähettäjäsolmun” tunnistekenttä:

```
INT32U MCP_CAN::getCanId(void);
```

4.1.5 Sanoman lähettäminen

Sanomia voi lähettää seuraavalla komennolla:

```
CAN.sendMsgBuf(INT8U id, INT8U ext, INT8U len, data_buf);
```

id kuvaa tunnistekenttää.

ext kuvaa sanomakehystyyppiä. Standard Frame-formaatissa käytetään 0:aa, ja Extended Frame -formaatissa käytetään 1.

len kuvaa kehyksen pituutta.

dat_buf kuvaa sanomakehyksen sisällön. [8]

4.1.6 Sanoman vastaanottaminen

Seuraavalla komennolla vastaanotetaan sanoma:

```
CAN.readMsgBuf(unsigned char len, unsigned char buf);
```

Tilanteessa, jossa maskit ja suodattimet on asetettu, tämä komento vastaanottaa vain niiden raameissa rajattuja sanomia. [8]

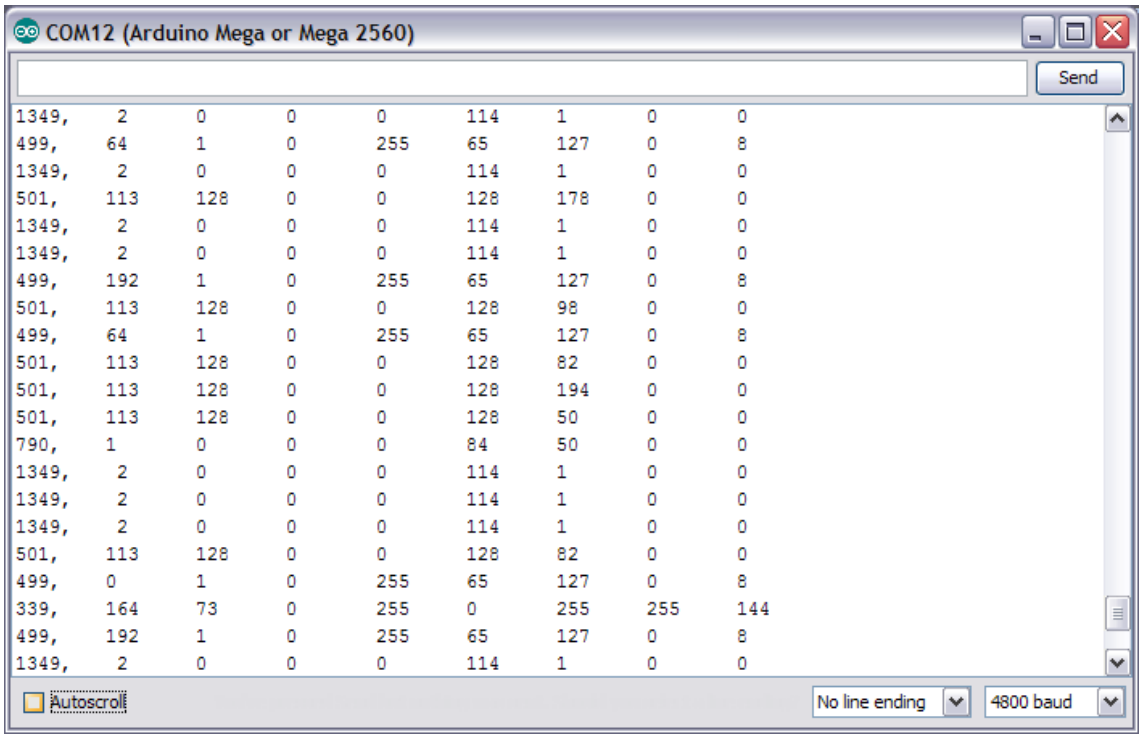
5 Suunittelutyön toteutus ja tulokset

Työ aloitettiin käymällä läpi erilaisia konsepteja, miten ohjauspaneelikonseptia voidaan toteuttaa. Lähtökohtana oli löytää mikrokontrollerialusta, jolla voidaan lukea CAN-väylässä kulkeva tieto. Vaatimuksena oli myös mikrokontrollerin yksinkertainen ohjelmointi, jotta sillä voitaisiin lähettää komentoja väylälle. Taustatyön perusteella valittiin Arduino-kehitysalustaan perustuvia mikrokontrollereita. Kosketusnäytön valinnassa otettiin huomioon yhteensopivuus Arduino-mikrokontrollerin kanssa sekä yksinkertainen kehitysympäristö, jolla voidaan luoda yksinkertaisia graafisia luonnoksia.

Arduino-alustalle valmistetut piirilevyt ja laajennuskortit ovat helposti kytkettävissä keskenään niihin hyvin sijoitutettujen I/O-liitäntäpinnien ansiosta. CAN-lisälaite asennetaan Arduino Megassa vastaavien pinnien päälle. Arduino Megassa on valmiiksi ohjelmoitu SPI-kirjasto, jolla kommunikointi CAN-lisälaitteen kanssa tapahtuu ilman lisätoimia.

Kosketusnäyön kanssa tarvitaan Arduino Display Shield -näyttöadapteri, jossa on I/O-liitäntäpinnit. Näyttö tarvitsee myös erillisen USB-adapterin. USB-adapteria käytetään tietokoneen kanssa, kun näytölle halutaan asentaa sovelluksia sovellusohjelmalla.

Mikrokontrolleri kytketään ajoneuvon CAN-väylään, sekä tietokoneeseen USB-väylän kautta. Arduino IDE:n kautta ladataan haluttu koodikirjasto mikrokontrolleriin. Kun halutaan lukea tietoa väylästä voidaan pohjana käyttää receiv_check-ohjelmakoodiesimerkkiä (liite 1). IDE:ssä avataan terminaali-ikkuna, josta näkee saapuvaa dataa. Receiv_check-ohjelmakoodilla saapuva tietovirtaa näyttää pelkästään tietokentän, eikä sisällä esimerkiksi tunnistekentän, jolla voidaan tunnistaa sanomien lähettämät solmut. Saadakseen selvitettyä tunnistekenttää, pitää määrittää komennolla ”INTU32U canId = 0x000” parametrit, joiden mukaan laite hakee sanomien sisältämät tunnistekentät. Tässä tapauksessa ”0x000” tarkoittaa 11-bittistä tunnistekenttää. Komennolla ”canId = CAN.getCANId ();”,(liite 2) mikrokontrolleri tulostaa terminaali-ikkunaan (kuva 17) sanomat sekä niiden tunnistekentät. Nyt voidaan tarkastella saapuvaa dataa sekä tunnistaa eri sanomat niiden tunnistekenttien perusteella.



Kuva 17. Kuvakaappaus Arduino IDE -terminaali-ikkunasta

CAN-väylällä liikkuu suuria määriä dataa ja saapuvaa tietovirtaa täytyy rajata, jotta löydetään halutut sanomat. Rajauksessa voidaan käyttää maski- ja suodatinrekisteriä, mutta jos haluaa rajata tietoliikennettä vain yhden sanoman selvittämiseksi voi siinä tapauksessa käyttää ”if” -ehtolausetta. Ehtolause perustuu tietokentän rajaukseen, jossa sanoma rajataan bittitavujen mukaan. Ehtolause voi esimerkiksi olla ”if (buf[0] == 192 && buf[1] == 91])”. Näin saapuvia sanomia on rajattu tietokentän kahden bittitavun mukaan ”192, 91”.

Rajauksen perusteella tarkasteltuja yksittäisiä sanomia voidaan käyttää omien komentojen lähettämiseen. ”Can.sendMsgBuf(INT8U id, INT8U ext, INT8U len, data buf);” -komentoon sijoitetaan vaadittavat parametrit ja haluttu sanomasisältö.

Ulkopuolisella mikrokontrollerilla ei pystytty kuitenkaan onnistuneesti lähettämään komentoa, jolla olisi voinut esimerkiksi käynnistää auton pyyhkijät. Jotta sellainen toiminto olisi mahdollinen, vaatisi se, että auton pyyhkijöiden oma ohjainlaite on kytketty pois CAN-väylästä.

6 Yhteenveto

Lähtökohtana oli rakentaa ohjainlaite, jolla pystyisi lukemaan ajoneuvon CAN-väylässä liikkuvaa tietoliikennettä sekä ohjaamaan ajoneuvon laitteita ja muita erilaisia toimintoja. Insinöörityölle alunperin asetettuihin tavoitteisiin päästiin vain osittain. Tavoitteena oli toteuttaa MoTeC moottorinohjausyksikköön yhdistetty kosketusnäytöllinen ohjauspaneeli, jolla pystyttäisiin ohjaamaan ajoneuvon toimintoja. Aiheen vaativuuden sekä kiireisen aikataulun johdosta, työssä keskityttiin konseptinsuunnitteluun sekä ohjelmakoodin tarkasteluun.

Työssä tutustuttiin CAN-väylän ominaisuuksiin ja niihin liittyviin standardeihin sekä viestien kehysrakenteeseen. Työssä selvitettiin ohjauspaneelin toteuttamiseen tarvittavat komponentit, kuten mikrokontrollerit ja kosketusnäyttö sekä tutustuttiin niiden ohjelmistoympäristöihin.

Insinöörityön varsinainen työ perustui ohjelmakoodin tarkasteluun, CAN-väylän sarjaliikenteen lukeminen sekä komentojen lähettämisen CAN-väylään. Tähän tavoitteeseen päästiin vain osittain. Arduino CAN-mikrokontrollerilla pystyttiin onnistuneesti lukemaan tietoa väylästä sekä pystyttiin tunnistamaan eräiden ohjainlatteiden lähettämiä sanomia.

Ajan myötä ajoneuvoissa lisääntyneen elektroniikan sekä hybridi- ja sähköautojen kehityksen siivittämä, on CAN-väylätekniikasta tullut yksi ajoneuvoteollisuuden tärkeimpiä osa-alueita, jonka pohjalta uusia väylätekniikoita kehitetään koko ajan vaihtuviin vaatimuksiin.

Lähteet

1 Paret, Dominique. 2007. Multiplexed networks for embedded systems: CAN, LIN, safe-by-wire---. Hoboken, N. J. : John Wiley & Sons.

2 Alanen, Jarmo. 2000. CAN – ajoneuvojen ja koneiden sisäinen paikallisväylä. Verkkodokumentti. Oulun Ammattikorkeakoulu. <http://www.oamk.fi/~eeroko/Opetus/Ohjausjarjestelmat/CAN/CAN-perusteet_AlasenMateriaalia.pdf>. Luettu 19.4.2014.

3 Di Natale, Marco. 2012. Understanding and Using the Controller Area Network Communication Protocol: Theory and Practice. New York, NY: Springer New York.

4 Saha, Heikki. 2005. CAN-väylä. Verkkodokumentti. FLUID Finland 4/2005. <http://www.canopen.fi/artikkelit/CAN.pdf>. Luettu 20.4.2014.

5 Arduino Mega2560. 2015. Verkkodokumentti. Arduino LLC. <http://www.arduino.cc/en/Main/ArduinoBoardMega2560>. Luettu 23.4.2015.

6 Arduino Software. 2015. Verkkodokumentti. Arduino LLC. <http://www.arduino.cc/en/Main/Software>. Luettu 23.4.2015.

7 SeeedStudio CAN Bus Shiled. 2015. Verkkodokumentti. Seeed Technology Limited. <http://www.seeedstudio.com/wiki/CAN-BUS_Shield>. Luettu 13.4.2015.

8 Github SeeedStudio CAN_BUS_SHIELD. 2015. Verkkodokumentti. GitHub, Inc. <https://github.com/Seeed-Studio/CAN_BUS_Shield> Luettu 27.4.2015.

9 4D Systems Intelligent Display Module. 2015. Verkkodokumentti. 4D Systems Pty Ltd. <http://www.4dsystems.com.au/product/uLCD_70DT/> Luettu 13.4.2015.

10 Topology Aspects of a High-Speed CAN bus. 2009. Verkkodokumentti. Semi-conductor Components Industries, LLC. <http://www.onsemi.com/pub_link/Collateral/AND8376-D.PDF>. Luettu 21.4.2014.

Liite 1. Ohjelmakoodikirjasto, receiv_check

```
// demo: CAN-BUS Shield, receive data with check mode
// send data coming to fast, such as less than 10ms, you can use this way
// loovee, 2014-6-13


#include <SPI.h>
#include "mcp_can.h"


unsigned char Flag_Recv = 0;
unsigned char len = 0;
unsigned char buf[8];
char str[20];


MCP_CAN CAN(10);                                    // Set CS to pin 10

void setup()
{
    Serial.begin(115200);

START_INIT:

    if(CAN_OK == CAN.begin(CAN_500KBPS))                    // init can bus : baudrate = 500k
    {
        Serial.println("CAN BUS Shield init ok!");
    }
    else
    {
        Serial.println("CAN BUS Shield init fail");
        Serial.println("Init CAN BUS Shield again");
        delay(100);
        goto START_INIT;
    }
}


void loop()
{
    if(CAN_MSGAVAIL == CAN.checkReceive())            // check if data coming
    {
        CAN.readMsgBuf(&len, buf);    // read data,  len: data length, buf: data buf

        for(int i = 0; i<len; i++)    // print the data
        {
            Serial.print(buf[i]);Serial.print("\t");
        }
        Serial.println();
    }
}

/*********************************************************************************************************
  END FILE
*********************************************************************************************************/
```

Liite 2. Ohjelmakoodi, send_message

```
// demo: CAN-BUS Shield, send data
#include <mcp_can.h>
#include <SPI.h>

MCP_CAN CAN(10);                                    // Set CS to pin 10

void setup()
{
    Serial.begin(115200);

START_INIT:

    if(CAN_OK == CAN.begin(CAN_500KBPS))                    // init can bus : baudrate = 500k
    {
        Serial.println("CAN BUS Shield init ok!");
    }
    else
    {
        Serial.println("CAN BUS Shield init fail");
        Serial.println("Init CAN BUS Shield again");
        delay(100);
        goto START_INIT;
    }
}

unsigned char stmp[8] = {0, 1, 2, 3, 4, 5, 6, 7};
void loop()
{
    // send data:  id = 0x00, standrad flame, data len = 8, stmp: data buf
    CAN.sendMsgBuf(0x00, 0, 8, stmp);
    delay(100);                       // send data per 100ms
}

/*********************************************************************************************************
  END FILE
*********************************************************************************************************/
```